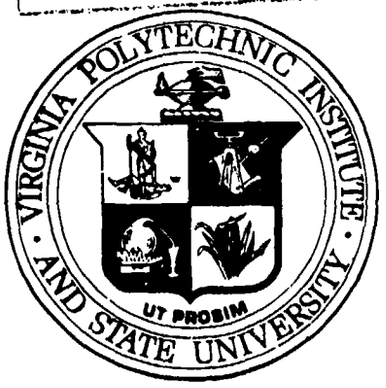
LEVEL II

10






# Virginia Polytechnic Institute and State University

Computer Science
Industrial Engineering and Operations Research
BLACKSBURG, VIRGINIA 24061

CSIE-81-4

October 1981

# A TWO-DIMENSIONAL CORE GRAPHICS SYSTEM FOR RESEARCH IN HUMAN-COMPUTER INTERFACES

Roger W. Ehrich

TECHNICAL REPORT

Prepared for
Engineering Psychology Programs, Office of Naval Research
ONR Contract Number N00014-81-K-0143
Work Unit Number NRSRO-101

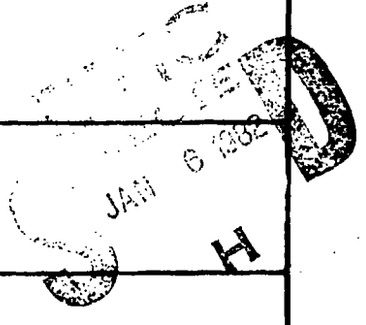
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|
| 1. REPORT NUMBER: CSIE-81-4 | 2. GOVT ACCESSION NO.: AD-A109280 |
| 3. RECIPIENT'S CATALOG NUMBER | |
| 4. TITLE (and Subtitle): A TWO-DIMENSIONAL CORE GRAPHICS SYSTEM FOR RESEARCH IN HUMAN-COMPUTER INTERFACES | 5. TYPE OF REPORT & PERIOD COVERED: Technical Report |
| | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s): Roger W. Ehrich | 8. CONTRACT OR GRANT NUMBER(s): N00014-81-K-0143 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS: Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061 | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: NRSRO-101 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS: Office of Naval Research, Code 455, 800 North Quincy Street, Arlington, VA 22217 | 12. REPORT DATE: October 1981 |
| | 13. NUMBER OF PAGES: 57 |
| 14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) | 15. SECURITY CLASS. (of this report): Unclassified |
| | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)




18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

graphics, standard, CORE, human factors

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Graphics is an important requirement for human-computer communication, and in order to construct good interfaces and conduct behavioral experiments, a flexible and reliable graphics package is required. Such a package must be extensible, and because of the dynamic nature of facilities, it must be quickly adaptable to new device types. This report describes a limited implementation of the ACM/SIGGRAPH Core System that will be the vehicle for the graphics aspects of the research program at Virginia Tech in human factors in computation.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

412524

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

## ACKNOWLEDGEMENTS

This research was supported in part by the Office of Naval Research and ONR Contract Number N00014-81-0143, and Work Unit Number NRSRO-101. The effort was supported by the Engineering Psychology Programs, Office of Naval Research, under the technical direction of Dr. John J. O'Hare.

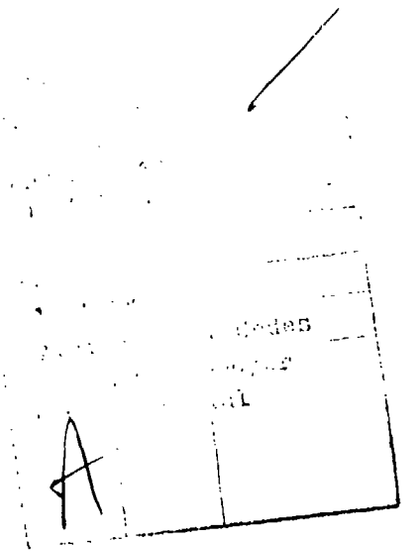


TABLE OF CONTENTS

# INTRODUCTION

Graphics is an important mechanism for implementing human-computer interfaces, and in order to conduct a significant research program that deals primarily with such interfaces, a quality graphics package is essential. While commercial packages are available, these are not easily modified and extended to support the particular needs of the research we are undertaking. Since the new ACM/SIGGRAPH standard is the only true standard, and since work has been under way for some time on an implementation of part of that standard, it was the obvious choice.

The CORE Graphics System was proposed by the Graphics Standards Planning Committee (GSPC) of ACM SIGGRAPH (Special Interest Group for Graphics). The CORE system is presently under investigation for possible acceptance as a standard graphics package by ANSI (American National Standards Institute), and by ISO (International Standards Organization).

Discussions of this implementation of the CORE graphics package began in January 1979, and implementation began a year later when the first VAX was purchased. The initial goals of this implementation were to provide a flexible graphics package for VAX users, while at the same time investigating the implementation problems and solutions for a subset of the CORE system. The full CORE System has not implemented; instead, the two-dimensional, no input, buffered output level has been developed. The retained segment capability of the buffered level has been extended to allow saving of segments as permanent files for later retrieval and reuse.

In this document, the CORE graphics system is described from a user's viewpoint by providing functional definitions of the CORE routines and examples showing use of the system.

The reference document for the CORE Graphics Standard is:

Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee, Part II: General Methodology and the Proposed Standard," Computer Graphics Quarterly Report of Siggraph-ACM, Vol. 13, No. 3, August 1979.

This report can be purchased for $12.00 (ACM Members) or $16.00 (ACM nonmembers) from:

ASSOCIATION FOR COMPUTING MACHINERY
1133 Avenue of the Americas
New York, New York 10036

The CORE System is not an applications package but a flexible control environment for the construction of graphics systems. Most important is the philosophy that any image producible on one display surface ought to be reproducible identically on any other, up to the limits of hardware capability. In part because of the richness of the capabilities of the devices the CORE System is intended to support, it is relatively sophisticated in concept and implementation. At this time of writing it consists of 211 modules and 8600 lines of code.

The CORE System discourages mixing user graphics with CORE graphics (for example, writing to a display without using the text primitive) because doing so eliminates portability and device independence of application code and because internal device states may be altered without the knowledge of the system. However, since there are clearly situations where special device features need to be utilized to obtain special goals, such as high speed operation, a

discussion is given of the problem of interacting with the CORE System.

The implementation described here does not support graphic input or the region fill algorithms available on raster graphics devices. Furthermore, the current package does not support dynamic graphics for real-time motion. Extensions will be added to the package to support these features as need arises and as suitable hardware becomes available. The effort required to interface new devices to the package varies with device complexity, and for a simple vector graphics terminal, a man-week would be sufficient.

The implementation described here is, in part, from the Masters Project of Alan D. Carwile in the Department of Computer Science at Virginia Tech. It was supervised by R. W. Ehrich who is continuing its development.

## FUNCTIONAL DESCRIPTIONS OF USER CALLABLE ROUTINES

To simplify use of the package from languages other than FORTRAN, only standard datatypes of FORTRAN 1977 were used throughout the implemencation. These data types are easily matched in most major languages, including PL/I and PASCAL. Three basic data types were used -- INTEGER, REAL, and CHARACTER. In the subroutine summaries that follow, the correct data type is indicated by the first letter of the argument according to accepted conventions. That is, if the first letter is between I-N inclusive, the argument is of INTEGER type. All other arguments are of REAL data type unless the summary contains an explicit reference to an argument being of type CHARACTER. Only four of the user callable subroutines contain arguments of type CHARACTER. These are Text, Inquire_text_extent, Save_retained_segment, and Retrieve_retained_segment. Any argument which is an array is indicated by appending the string 'array' to the argument name.

The subroutines are presented in sections according to function. These sections are:

Output Primitives -- These subroutines are used to create graphical entities for display.

Output Primitive Attributes -- These subroutines are used to alter the graphical entities (hereafter called 'primitives') by affecting such attributes as color, intensity, character size, and marker symbol.

Picture Segmentation -- A segment is a set of output primitives manipulated as a unit. These subroutines are used to create and delete segments.

Segment Attributes -- The picture may be modified by altering the visibility of retained segments.

Viewing Controls -- These subroutines are used to alter the placement and orientation of segments in the picture.

Control -- These subroutines are used to initialize or terminate use of the graphics package. Clipping, batching of updates, and immediate visibility are also controlled by routines in this section.

Special features -- These are internal CORE system subroutines and additions to the core system that are designed to facilitate interactions with the system.

The following syntax is used to describe each subroutine:

```
Subroutine_name (argument_one, argument_two)
```

Note that the subroutine name need not be capitalized in the fashion shown. All compilers on the VAX ignore case except in character literals.

## 2.1 OUTPUT PRIMITIVES

The output primitives are the basic building blocks of the CORE Graphics System. There are three basic types of output primitives -- lines, text, and markers. Each output primitive has a set of attributes which affect the precise form the output primitive takes. These attributes are listed in this section but are explained in Section 2.2.

The locations specified in calls to output primitive functions are given in world coordinates. The world coordinate system is any arbitrary coordinate system which the user selects. For example, for a mapping application the world coordinate system might be measured in miles or kilometers. For a physics application the world coordinate system could be in angstroms or millimeters. The world coordinate system is mapped to the display area of the terminal through a series of transformations described in the Viewing Controls section.

In order to use the output primitives, one must be familiar with the concept of current position. The current position is a location in world coordinates which is maintained by the package and which is altered after the display of output primitives. The display of a primitive is dependent on the current position for line and text output, while the display of marker output is independent of this position. The current position is updated to the last position of the output primitive after the subroutine is

called for line and marker primitives. Current position after a text primitive is the same as before the text primitive was called. The current position is maintained so that an application program may easily concatenate output primitives.

The output primitives come in several varieties. The user may generate individual lines with the Line subroutines, or sets of connected lines with the Polyline subroutines. One can generate individual markers or sets of markers in the same fashion with the Marker and Polymarker subroutines. Character output may be generated with the Text subroutine.

Absolute and relative positioning of primitives is provided. Each subroutine has an absolute and a relative counterpart. Relative positions are converted internally to absolute positions for eventual display. Thus a relative position (dx,dy) of (3,5) when the current position (CP) is at (20,25) would be translated by the package to an absolute position in world coordinates of (23,30).

### 2.1.1 Move abs 2 (x, y)

The current position (CPx, CPy) is updated to the position (x, y) in world coordinates. No visible picture change occurs due to this primitive. Its purpose is to alter the position of future output primitives.

### 2.1.2 Move rel 2 (dx, dy)

The current position (CPx, CPy) is updated to the position (CPx+dx, CPy+dy) in world coordinates.

### 2.1.3 Inquire current position 2 (x, y)

The current position is copied to the variables x and y.

### 2.1.4 Line abs 2 (x, y)

A line is drawn from (CPx, CPy) to the position (x, y) in world coordinates. The line's appearance is affected by the color, intensity, linestyle, linewidth, and pen attributes. If the position (x, y) is the same as the current position, a point is made at that location. After this primitive is executed, the current position is updated to the point (x, y).

### 2.1.5 Line rel 2 (dx, dy)

A line is drawn from (CPx, CPy) to the position (CPx+dx, CPy+dy) in world coordinates. The line's appearance is affected by the color, intensity, linestyle, linewidth, and pen attributes. If the offset (dx,dy) is (0,0), a point is made at the current position. After this primitive is executed, the current position is updated to the point (CPx+dx, CPy+dy).

### 2.1.6 Polyline abs 2 (xarray, yarray, n)

A connected series of lines is generated from the current position to (xarray(1), yarray(1)) to (xarray(2), yarray(2)) to ... (xarray(n), yarray(n)), where these points are defined in world coordinates. The connected lines are subject to the color, intensity, linestyle, linewidth, and pen attributes. If all of the points are coincident with (CPx, CPy), then a point is drawn at that location. The current position is updated to (xarray(n), yarray(n)) after this function. The number of points one may give in the x and y arrays is unbounded.

### 2.1.7 Polyline rel 2 (dxarray, dyarray, n)

A series of connected lines is generated beginning at the current position and proceeding through the indicated points. The first point drawn to is (CPx + dxarray(1), CPy + dyarray(1)). The next point is (CPx + dxarray(1) + dxarray(2), CPy + dyarray(1) + dyarray(2)). This pattern continues through the n-th position. The series of lines is subject to the current attributes of color, intensity, linestyle, linewidth, and pen. If all the offsets in dxarray and dyarray are zero, then a point is generated at the current position. After this function the current position is updated to the last point. The number of points one may give in the dx and dy arrays is unbounded.

### 2.1.8 Text (char string)

A series of text elements is generated with this function. The manifestation of the characters is subject to the current attribute settings for color, intensity, pen, character size, character spacing, character justification, character precision, character path, character rotation (charup), and character font. These attributes combine to affect the way the text is positioned relative to the current position. The current position after a Text function call is the same as it was prior to the call. If it is necessary to append other primitives to the end of the string, use the Inquire_text_extent_2 function below. The string may be of any length. Note that the argument is a CHARACTER*(*) variable on the VAX system.

Conceptually, consider every upper case character to be enclosed by a square box called the character box. The location of each character box is computed in world coordinates, clipped, and transformed to the screen system, together with all the character graphics within the box. Descenders and a few special characters may extend above or below the character box, but no characters are wider than the character box. The locations of the character boxes are affected by the charsize, charpath, charspace, and charjust attributes.

### 2.1.9 Inquire text extent 2 (char string, dx, dy)

This function is provided to allow concatenation of text in successive calls. The function takes a CHARACTER*(*) variable as its argument and computes an offset in world coordinates. The offset indicates the distance from the present CP to a new CP'. By moving to the new CP' and then writing more text, the user can cause concatenation to text which was output at the old CP. This function takes into account the settings for character size, spacing, etc. The argument char_string may be of any length and is a CHARACTER*(*) variable.

### 2.1.10 Marker abs 2 (x, y)

This function generates a marker at the specified location in world coordinates. The current position is updated to (x, y) after this function. A marker is generated at the proper location, but is not scaled, rotated, or translated by the viewing transformation or any other transformation in the CORE system. Markers are designed to be always of the same size and orientation. That is, they are unrotated and of the smallest size feasible for the display device. Markers are not restricted to the symbols in the set of recognizable text strings. However, since most display devices do not possess special symbols other than text characters, the markers available will normally be standard characters. (See the marker symbol attribute in the following section.)

### 2.1.11 Marker rel 2 (dx, dy)

A marker is generated at the offset specified from the current position, and then the current position is updated to that position (CPx + dx, CPy + dy). The marker symbol attribute determines which marker will be generated.

### 2.1.12 Polymarker abs 2 (xarray, yarray, n)

A series of markers is generated at the indicated locations in world coordinates. The current position is then updated to the last position in the x and y arrays. The number of points allowed is unbounded. The marker symbol attribute determines the symbol used for all locations.

### 2.1.13 Polymarker rel 2 (dxarray, dyarray, n)

A series of markers is generated at the indicated points. Note that each offset specified in the dx and dy arrays is relative to the preceding point, not to the original current position (following the fashion of Polyline_rel_2). Current position is then updated to the last point. Any number of markers may be generated in a single call.

## 2.2 OUTPUT PRIMITIVE ATTRIBUTES

The output primitives of the CORE system allow the generation of lines, text, and markers. The output primitives are very flexible due to the provision of output primitive attributes. Each output primitive is affected by a set of attributes. Lines are affected by the current settings for color, intensity, linestyle, linewidth, and pen. Markers are affected by the settings of marker symbol, color, intensity, and pen. Text is affected by color, intensity, pen, font, character size, character up (rotation), character precision, character justification, character path, and character spacing.

With such a large set of attributes, it is essential that each be easily controlled by the user. The CORE system therefore provides SET and INQUIRE functions for each of these attributes. In the subsections which follow, only the SET functions are listed. In every case, a corresponding INQUIRE function also exists.

### 2.2.1 Set color (icolor)
### Inquire color (icolor)

The color attribute is changed to the specified value for all subsequently created output primitives. Color must be a positive integer. The color attribute affects all types of output primitives. The actual color indicated by a particular value is not set in this implementation. The mechanism for setting up a color lookup table is described

in the CORE System Raster Extensions Report. The subroutines described therein are not implemented at the current time. As devices supporting color are added, these functions will be provided. The default color is 1.

### 2.2.2 Set intensity (rintensity)<br>Inquire intensity (rintensity)

The intensity attribute is a real number between 0.0 and 1.0 inclusive. The value of 0.0 represents minimum possible intensity, while the value of 1.0 denotes maximum possible intensity. The default value is 0.5. The intensity attribute affects all classes of output primitives subsequently generated. As with the color attribute, intensity is defined in the CORE System Raster Extensions Report. Therein, the intensity attribute is described as interacting with the color attribute to define a particular graphic result. The way in which interaction occurs is not defined nor implemented at this time. As further experience with raster type devices is gained, the interaction should be analyzed and functions to work with intensity should be added.

### 2.2.3 Set linestyle (ilinestyle)<br>Inquire linestyle (ilinestyle)

The linestyle attribute is defined as a positive integer. This implementation supports the definition of linestyles 1-5 as solid, coarse dashed, medium dashed, dotted, and alternating dashed. This attribute affects line

output primitives only. At the present time, only the TEK4012 device drivers do not differentiate between the five linestyles.

### 2.2.4 Set linewidth (rlinewidth)
### Inquire linewidth (rlinewidth)

The linewidth attribute is defined as a real number between 0.0 and 1.0 inclusive. The value indicates a percentage of the ndc-space extent (see Section 2.5). Thus a value of 0.3 would imply a line whose width is three-tenths the current ndc-space extent. (If x-extent and y-extent are not equal, the minimum is used.) The linewidth attribute affects all subsequently created line output primitives. The default linewidth attribute is 0.0 which indicates the minimum width line possible as opposed to a line of no width.

### 2.2.5 Set pen (ipen)
### Inquire pen (ipen)

The pen attribute is defined as a non-negative integer. If its value is 0, it has no effect on any output primitive. If its value is greater than zero, the current display device uses this pen number when displaying all classes of output primitives. In this situation, color, intensity, linestyle, and linewidth are ignored, and default values are used. The default pen value is zero. The pen attribute is most often used for plotters, and 4 positive pen values are defined for the HP7221.

### 2.2.6 Set font (ifont)
### Inquire font (ifont)

The CORE system allows the definition and use of multiple fonts. The font attribute is defined as a positive integer whose default value is 1. Only one software font is currently available for stroke precision text. This implementation is setup to handle only mono-spaced fonts. Proportional spacing is not provided. Font number 1 results in use of the standard ASCII character set for string and character precision.

### 2.2.7 Set charsize (rwidth, rheight)
### Inquire charsize (rwidth, rheight)

Character size is defined with two components, width and height. Both must be positive real numbers and are defined in world coordinates. The default character size has a width of 0.01 and a height of 0.01, allowing 100 characters in each direction with the default window and viewport.

### 2.2.8 Set charup 2 (xpart, ypart)
### Inquire charup 2 (xpart, ypart)

The charup attribute defines the rotation of each character in a text primitive. It is specified as a vector having x and y components. When the user sets a rotation with, say an x component of 3.0 and a y component of 4.0, these values are normalized. The normalized values are maintained by the system, such that a call to Inquire_charup_2 (xpart, ypart) would find xpart equal to

0.6 and ypart equal to 0.8. When setting the components of the charup vector, use any real values, but be sure that at least one is non-zero.

The vector so defined represents the upward direction of each character created by subsequent text output primitives. To see what rotation a given charup value will give, draw the vector out from an arbitrary point. Then position a box corresponding to the current charsize values around the vector, with the height component along the direction of the vector. Then position a character in this character box.

### 2.2.9 Set charpath (ivalue)<br>Inquire charpath (ivalue)

The character path defines the direction in which text proceeds if more than one character is output. The possible values are: 1 - towards the right (default), 2 - towards the left, 3 - upward, and 4 - downward.

### 2.2.10 Set charspace (rspace)<br>Inquire charspace (rspace)

The character spacing attribute controls the amount of space to be added between characters in a text output primitive. This attribute may be any real number -- positive, negative, or zero. Its default value is 0.0 which indicates no additional spacing. The font designer must allow a "reasonable" amount of space for inter-character gaps, so that a value of zero for charspace gives a

reasonable looking result. Values greater than zero add in space, while negative values take away space and eventually cause text to overlap or even to proceed backwards. The value of charspace indicates the fraction of charsize (width if charpath is 1 or 2, height if charpath is 3 or 4) to be added in. Thus, a value of 0.5 indicates that additional space corresponding to half a character in size should be added to spread out the characters.

### 2.2.11 Set charjust (ijustx, ijusty)<br>Inquire charjust (ijustx, ijusty)

The positioning of a text string relative to the current position is critical in some applications. For this reason the character justification attribute is provided. It consists of two parts -- a horizontal justification (ijustx) and a vertical justification (ijusty). Each may have a value of 0, 1, 2, or 3. Values 1, 2, and 3 represent left, center, and right justification in the horizontal direction, and bottom, center, and top justification in the vertical direction, respectively.

As an example, if this subroutine is called with arguments (1,1), subsequent text will be positioned so that the lower left corner of the string is at the current position. The justification values indicate the point in the string extent (not just the first character) which is placed at the current position. Thus an entire string can be centered in the x direction by specifying a horizontal justification value of two.

The default values of (0,0) allow the text to be located relative to CP in a fashion dependent on the current character path. For charpath of 1 (rightwards), the string baseline (bottoms of upper case characters) passes through the CP, and the string will grow out from the CP towards the right. For charpath of 2 (leftwards), the string baseline again passes through the CP, but the string extends this time to the left. For charpath of 3 (upwards), there is horizontal centering and upward growth, and for charpath of 4 (downwards), there is horizontal centering and downward growth.

### 2.2.12 Set charprecision (ivalue)<br>Inquire charprecision (ivalue)

Due to the complexity of the text output primitive, it is sometimes too expensive to generate text with all of the attributes previously mentioned. Therefore, one may specify a precision with which the attributes are to be followed. Character precision may take the values 1, 2, and 3. With character precision equal to 3 (stroke precision) the text is displayed as connected lines drawn according to all text attributes.

A value of 1 (string precision) means that the hardware character generator must be used to output the string. In string precision, CORE determines where the stroke precision text string would be positioned and then centers the text string produced by the hardware in the

center of this text field. The hardware character size is matched as closely as possible to the value of the charsize attribute.

An intermediate level with value 2 (character precision) is also provided. With this precision, each character is positioned individually and then generated by the hardware character generator. In this case, CORE determines where each character of stroke precision text would be positioned and then centers each text character produced by the hardware in the center of that field. Again, the hardware character size is matched as closely as possible by the charsize attribute. The default character precision is 1 -- string precision.

### 2.2.13 Set marker symbol (ivalue)<br>Inquire marker symbol (ivalue)

The marker symbol attribute determines the particular marker to be output for marker output primitives. Its value may be any integer from one to ten, inclusive. These are defined as follows:

1 - period
2 - plus
3 - asterisk
4 - capital O
5 - capital X
6 - capital A
7 - capital B
8 - capital C
9 - capital D
10 - capital E

## 2.3 Picture Segmentation

The CORE System provides a segmented data structure for grouping of primitives. A segment is a logical graphical entity consisting of output primitives and attribute modifications. One or more segments define the current "picture." There is no capability for defining a segment in terms of other segments.

There are two types of segments in the CORE System -- temporary and retained. Retained segments provide a capability for selectively modifying the picture by deleting segments and/or making them temporarily invisible. This capability does not exist for temporary segments. Temporary segments can be deleted only by calling New_frame to clear the display surface.

This implementation provides both temporary and retained segments, as well as an extension which allows a retained segment to be saved as a permanent file or to be retrieved from a permanent file. However, it does not provide the full range of segment attributes as discussed in the GSPC CORE report. This implementation only provides the visibility segment attribute. It does not provide the highlighting, detectability, or image transformation type attributes of segments.

### 2.3.1 Create_temporary_segment

A temporary segment is begun by a call to this function and ended with a call to Close_temporary_segment. Output

primitives are in error unless within the scope of an open segment, either temporary or retained. A call to this function freezes the viewing parameters (window, viewport, clipping, etc.) so that all output primitives in this segment will be generated according to the same view. While the segment is open, calls to set output primitive attributes and to actually generate output primitives are most common. However, INQUIRE subroutines may be used to check any CORE attribute or control variable.

When the output primitives are called they may or may not become visible immediately, depending on several controls. These are described more completely in the Control section, but are outlined here. The temporary segment may or may not be visible depending on the system-wide visibility attribute. If the current value is 0 (meaning invisible), calls to the output primitives have no visible effect. If the value is 1 (visible), the output primitives will be shown. If the immediate visibility attribute is set on (1), each primitive is made visible immediately. Otherwise, the system buffers up the display code for generating primitives until either the buffer is full or the Make_picture_current function is invoked.

### 2.3.2 Close temporary segment

The currently open temporary segment becomes closed, thus preventing calls to the output primitives until another call to Create_temporary_segment or Create_retained_segment.

Since temporary segments do not possess a segment attribute of visibility, their view cannot be manipulated except by deletion. A New_frame action causes the deletion of all temporary segments.

### 2.3.3 Inquire open temporary segment (iopen)

The argument 'iopen' is set to 1 if there is a temporary segment currently open. Otherwise, it is set to 0.

### 2.3.4 Create retained segment (name)

This function opens a retained segment, allowing output primitives to be generated. The current viewing parameters (window, viewport, clipping, etc.) are frozen until a subsequent Close_retained_segment operation. A retained segment is similar to a temporary segment in that the segment takes its own initial visibility from the current system-wide visibility attribute. However, each retained segment's visibility flag can be altered after creation. As each output primitive in an open retained segment is called, the picture on the display surface will be updated depending on the immediate visibility attribute in the same fashion as for temporary segments. A retained segment's name must be an integer between 1 and 32767, inclusive.

2.3.5 Close retained segment The currently open retained segment is closed making further calls to output primitives invalid, unless another segment is first created. The just

closed retained segment can only be manipulated in one way after it is closed. Its segment visibility attribute may be altered.

### 2.3.6 Inquire open retained segment (name)

The argument name is set to the name of the currently open retained segment. If no retained segment is open, name is set to zero.

### 2.3.7 Inquire retained segment names (isize, namearray, nsegments)

This function is used to get a list of the names of all retained segments. The list is copied into namearray in ascending order by segment name and the number of segments currently defined is copied into nsegments. Isize indicates the number of cells in namearray. If isize is less than the number of segments currently defined, only isize of them are copied, but the actual number of segments is copied to nsegments.

### 2.3.8 Delete retained segment (name)

After a retained segment is created, defined, and closed, it does not go away. Retained segments exist until either explicitly deleted or until Terminate_core is called. The argument is used to tell which retained segment should be deleted. The deletion will occur immediately unless a batch of updates is in progress. If a batch is in progress, the name is not available for reuse until an invocation of End_batch_of_updates.

### 2.3.9 Delete all retained segments

This function is available so that the user can delete all retained segments at one time without knowing each segment's name. The deletions will occur immediately unless a batch of updates is in progress. If a batch is in progress, the names are not available for reuse until an invocation of End batch of updates.

### 2.3.10 Rename retained segment (name, newname)

The existing retained segment is renamed from name to newname. The old name is immediately available for reuse.

### 2.3.11 Save retained segment (name, filename)

The existing retained segment called name is preserved in a permanent file called filename. If that file already exists, it will be overwritten. Otherwise it will be created. The content of the created file is readable ascii code which represents the image of the output primitives in the segment. No visible picture changes occur as the result of this function. The filename argument is a CHARACTER*(*) variable.

### 2.3.12 Retrieve retained segment (name, filename)

This function fetches the permanent file called filename and retrieves the definition of the retained segment described in it, thus creating a new retained segment for use in the current program. The specified

segment name is assigned to the new segment. The visibility of the retrieved segment is taken from the current visibility of the CORE system just as occurs for the Create_retained_segment function. Retrieve_retained_segment may not be invoked while a segment is already open. A visible picture change will occur if the segment is visible. The filename argument is a CHARACTER*(*) variable.

## 2.4 Segment Attributes

A segment attribute is a characteristic of the segment which can be modified after the segment has been created, defined, and closed. In this implementation only one segment attribute is defined. Highlighting, detectability, and image transformation type are not provided. Visibility is provided. Each time a segment is created, its visibility is copied from the current setting for the system-wide visibility attribute. For temporary segments, visibility is fixed and unmodifiable after creation. However, SET and INQUIRE functions exist to manipulate the visibility of previously defined retained segments.

In the descriptions which follow, only the SET functions are discussed. However for each SET function a corresponding INQUIRE function exists, with comparable syntax.

### 2.4.1 Set visibility (ivalue)
### Inquire visibility (ivalue)

This function sets the system-wide visibility for segments which will be subsequently created. Ivalue must be either one or zero, with zero meaning invisible and one meaning visible. Existing segments, including the open segment are not affected by this call.

### 2.4.2 Set segment visibility (name, ivalue)
### Inquire segment visibility (name, ivalue)

The current visibility for the named retained segment is set to ivalue, where ivalue must be zero or one, as above. The visibility change will show up right away on the display surface unless a batch of updates is in progress. If a batch of updates is in progress, the change will not be noticeable until a call to End batch of updates.

### 2.4.3 Set visibilities (namearray, ivaluearray, n)
### Inquire visibilities (namearray, ivaluearray, n)

This function is provided so that several named segments can have their visibility attributes changed as a group. The altering of segment visibilities proceeds from the first to the n-th, with no check for duplicates in the namearray. Thus, if duplicates occur, the occurrence latest in the arrays will be in effect after this call. Visibility changes will show up right away on the display surface unless a batch of updates is in progress. If a batch of updates is in progress, the changes won't be noticeable until a call to End_batch_of_updates.

## 2.5 Viewing Controls

The coordinates used by the user in calls to the output primitives are defined in an arbitrary system called world coordinates. In order to generate a display, it is necessary that the coordinates be converted to screen units in a device-independent fashion. This section describes the subroutines which affect the transformation of world units to screen units.

The first transformation is the world transformation. This transformation is available to facilitate the building of a modelling system on top of the CORE system in which objects can be defined in arbitrary coordinate systems and then moved into the world system. It defaults to an identity transformation. The second transformation is the viewup transformation. This transformation allows the user to cause a rotation of the world coordinates about the origin of the world coordinate system.

Next comes the windowing transformation. The user specifies a window in world coordinates which determines the portion of his coordinate space which he wants to make visible on the display device. He also specifies a viewport in normalized device coordinates which determines the region of the display surface onto which the window is mapped. At this point, the image of the segment may (under user option) be clipped to the viewport region.

If the segment is a retained segment, its image is then stored to allow redisplay if visibility is changed later. The last two steps are the image and screen transformations. Image transformations are not provided in this implementation. The screen transformation maps the normalized device coordinates to screen coordinates.

The series of operations outlined above takes the coordinates used in the output primitive invocations and maps them to screen coordinates after clipping and storing. The screen coordinates are then used by the device specific drivers to generate the appropriate pictures. The user need not concern himself with the screen transformation since this is taken care of by the system automatically. However, the following functions allow one to alter the other operations to suit the application. Note that the SET functions below each have a comparable INQUIRE function.

### 2.5.1 Set ndc space 2 (width, height)<br>Inquire ndc space 2 (width, height)

This function may be used at most once per call to Initialize_core. If used, it must precede all calls to any of the following:

Create_temporary_segment

Create_retained_segment

Set_viewport_2

Inquire_viewport_2

This call specifies the range of normalized device coordinates that can be made visible on the display. Both values must be positive real numbers not greater than 1.0, but at least one must actually be 1.0. Invoking this function causes an implicit setting of the viewport to the range of (0.0, width, 0.0, height). The default range of ndc-space is (0.0, 1.0, 0.0, 1.0). With the default setting, the entire addressable area on the display is available. Changing the extents of ndc-space is useful for devices whose visible display areas are not square.

### 2.5.2 Set view up 2 (xpart, ypart)<br>Inquire view up 2 (xpart, ypart)

This function specifies the world coordinate up direction, and it should not be called while a segment is open. The vector derived from the x and y components supplied as arguments defines a rotation of the world coordinates under user control. The positive y-axis of the world system is rotated about the origin to coincide with the specified view_up vector. The components may be any real numbers, but at least one must be non-zero. The values for xpart and ypart are normalized such that using the INQUIRE function may return a different value from that originally specified in this function. The default view_up vector is (0.0, 1.0), giving no rotation.

### 2.5.3 Set window (xmin, xmax, ymin, ymax)
### Inquire window (xmin, xmax, ymin, ymax)

The arguments to Set window define a rectangle in world coordinates. The subroutine should not be called while a segment is open. The rectangle is mapped to the viewport rectangle defining the windowing transformation mentioned above. Thus, the area within the rectangle is the area which will be made visible on the display surface. The arguments may be any real numbers, provided that xmin is strictly less than xmax and ymin is strictly less than ymax. The default window is (0.0, 1.0, 0.0, 1.0).

### 2.5.4 Set viewport 2 (xmin, xmax, ymin, ymax)
### Inquire viewport 2 (xmin, xmax, ymin, ymax)

This function is used to define the region of normalized device coordinates to which the window will be mapped, and it should not be called while a segment is open. The arguments are real numbers between 0.0 and 1.0, inclusive. An added restriction is that the viewport must fit within the ranges of ndc-space as defined in a call, if one has occurred, to Set_ndc_space_2. The default viewport is (0.0, xsizendc, 0.0, ysizendc).

### 2.5.5 Set window clipping (ionoff)
### Inquire window clipping (ionoff)

This function controls whether the output primitives in subsequently created segments will be clipped to the window (or viewport equivalently), and it should not be called while a segment is open. The argument must be either 0 or

1, with 1 enabling clipping and 0 disabling clipping. Clipping is enabled by default.

### 2.5.6 Set world coordinate matrix 2 (realmatrix) Inquire world coordinate matrix (realmatrix)

This subroutine accepts a 3x3 array and copies its values in as the new world transformation. The 3x3 matrix must have (0.0, 0.0, 1.0) as its last column and must be invertible, or an error will be flagged. The INQUIRE subroutine for this function has a name longer than 31 characters. Due to the VAX's limitation of names to 31 characters, the corresponding inquiry function is Inquire_world_coordinate_matrix (realmatrix). The default world transformation is the identity transformation. Note: If calling from other than FORTRAN, some languages (e.g. PL/I) store two-dimensional arrays row by row, while FORTRAN stores them column by column.

### 2.5.7 Map world to ndc 2 (xworld, yworld, xndc, yndc)

This function accepts as input a location in the world coordinate system. It maps that location to ndc coordinates. If the point is out of the viewport and clipping is enabled, an error will be flagged. All arguments are real.

### 2.5.8 Map ndc to world 2 (xndc, yndc, xworld, yworld)

This function maps a location in ndc coordinates to the corresponding location in world coordinates. If the point

is outside the viewport, an error will be flagged. All arguments are real.

## 2.6 Control

These subroutines are used to control the general operation of the CORE system. Since this implementation supports only one device at any given time, the number of subroutines of this type is greatly reduced from the number described in the CORE System Report. These subroutines fall into three classes:

Initialization / Termination

Picture Change Control

Error Handling

### 2.6.1 Initialize core (level output, level input, level-dim)

This subroutine must be called before any other CORE function, so that transformations and attributes can be set to their default values. The three parameters are compared to pre-set values to see if the current version of the CORE system supports those levels. If not, an error is generated. This version supports level 2, or 'buffered', output which allows retained segments as well as temporary segments. Level 1 is supported for input, meaning no input functions. Level 1 is supported for dimension which indicates a two-dimensional (as opposed to a three-dimensional) implementation. This subroutine also checks what display device is in use and assures that its device driver is also initialized.

### 2.6.2 Terminate core

This subroutine cleans up any left over data structures and files and sets a flag to indicate that CORE is not initialized. A further call to Initialize_core is required before invoking any other CORE function.

### 2.6.3 Set immediate visibility (ionoff)<br>Inquire immediate visibility (ionoff)

This function is used to turn immediate visibility on (1) or off (0). When immediate visibility is on, each subroutine which generates graphic output automatically dumps the buffer before returning to the application program. When off, the system may delay the graphics result by buffering up the display code until the buffer either fills or until the user calls Make_picture_current or Set_immediate_visibility again to set it on. A corresponding Inquire_immediate_visibility function is provided. By default, immediate visibility is on.

### 2.6.4 Make picture current

This subroutine is useful when immediate visibility is off for efficiency reasons, but the picture must be made up to date. It causes the buffer to be sent but does not imply that batching of updates will be ended.

### 2.6.5 Begin batch of updates

When a user wishes to make many consecutive changes to the picture by deleting segments or making visible segments

invisible, one can surround the changes in a batch of updates. Depending on the class of display device, this can prevent disruption by numerous clears and redraws of visible segments. With a batch of updates in progress, deletions and visibility changes are deferred until an End_batch_of_updates.

### 2.6.6 End batch of updates

This function causes the picture to be made consistent with any segment deletions or visibility changes. However, it does not automatically send the buffer unless immediate visibility is on. If immediate visibility is off, follow a call to End_batch_of_updates with a call to Make_picture_current to assure the buffer is sent.

### 2.6.7 Inquire batch of updates (ionoff)

If a batch of updates is in progress, ionoff is set to one. Otherwise, it is set to zero.

### 2.6.8 New frame

This subroutine clears the screen of all non-retained segments and redraws all visible retained segments.

### 2.6.9 Report most recent error (idnumber, iseverity)

This function inquires the current values of two globals in the system and makes their values available to the application program. Idnumber indicates the number of the last error encountered. Iseverity indicates its

severity. If no error has occurred since the last call to this function, zeros are returned. For a list of errors, consult the Appendix.

### 2.6.10 Error_handler (idnumber, iseverity)

This function is not called by the user. When an error is recognized in the CORE system, this function is called from within the package. This function then passes the same arguments to the Log_error subroutine below. However, by compiling one's own Error_handler subroutine and linking it in with one's program, a user can override the default handler. For example, if a user wished to decide whether to continue execution or not, an Error_handler like that below could be written:

```
SUBROUTINE ERROR_HANDLER (IERR, ISEV)
CALL LOG_ERROR (IERR, ISEV)
PAUSE 'Core error'
RETURN
END
```

This subroutine would first use the Log_error subroutine to write a message to the CORE error unit, 24. Then the PAUSE statement in VAX FORTRAN would let the user have control. The PAUSE statement allows one to stop execution using the STOP command, continue execution using the CONTINUE command, or enter the symbolic debugger using the DEBUG command.

### 2.6.11 Log_error (idnumber, iseverity)

This subroutine looks up the idnumber in a table of error messages and writes a message to unit 24. A check is

made to ensure that the severity code matches the value associated with the idnumber. Consult the Appendix for a list of error messages by idnumber. This subroutine should be invoked only within an Error_handler.

### 2.6.12 Get error message (idnumber, iseverity, string)

This subroutine looks up the error message indicated by idnumber, returns the severity code, and copies the text of the message into string. Char_string should be set up as a CHARACTER*80 variable. This subroutine should only be invoked within an Error_handler.

## 2.7 Special features

Users who need to insert their own display instructions into the graphics stream, though discouraged from doing so, may use the following CORE internal subroutine.

### 2.7.1 Bufadd (iarray,n)

Bufadd inserts characters stored in an INTEGER*4 array into the graphics buffer and supervises their transmission to the display device. To ensure immediate transmission, if required, the user may use the Make_picture_current subroutine.

The present implementation does not provide for interactive input. Since terminals like the HP2648 require ENQ/ACK handshake protocols, there is in general no way to obtain terminal input while graphics is in progress to a non-slave device. In order that graphics in progress may be interrupted, the following subroutine is provided.

### 2.7.2 Inquire last character (ivalue)

When the display surface is not a slave surface, the most recent keyboard character received while graphics is in progress is saved, except for handshake return characters. The last character is cleared to null (0) by Initialize_core and by a call to Inquire_last_character.

Since display device identification is handled internally by the CORE system, a subroutine is provided to return the device type to the user.

### 2.7.3 Inquire device type (itype)

This subroutine returns the identification code of the current display device. At present the following values are defined:

1 - TEK4012/13

2 - HP2648

3 - HP7221 (slave)

# USING THE CORE GRAPHICS SYSTEM

This chapter presents several sample applications to indicate some of the ways the CORE system may be used. Only facilities implemented in this project are used in the examples. Two examples will be given. The first application deals with data plotting. This application is a static application, so retained segment options are unneeded. In the second application, a wheel is created in several 2-D views, giving the impression that the wheel is rotating about its axis. Finally, at the end of the chapter, the details for using the CORE system on the VAX are discussed in full.

## 3.1 A Static Application -- Data Plotting

For a beginner, the CORE system can be used very simply. There is no need to be concerned with batching of updates, segment visibility, or modelling transformations. For example, if the user wishes to plot a set of data as a graph, the CORE system defaults make the job easier:

```
C
C          MAIN PROGRAM AND SUBROUTINES
C          FOR THE SAMPLE PLOT
C
      DIMENSION X(100), Y(100)
      DO 50 I=1,100
           X(I)=I
50         Y(I)=I*I/100.0
      CALL INITIALIZE_CORE (1,1,1)
      CALL SET_WINDOW(-20.,120.,-20.,120.)
      CALL CREATE_TEMPORARY_SEGMENT
           CALL SET_CHARPRECISION(3)
           CALL SET_CHARSIZE(2.5,2.5)
C               X-AXIS LINE
           CALL MOVE_ABS_2(0.,0.)
```

```
            CALL LINE_ABS_2(100.,0.)
C                Y-AXIS LINE
            CALL MOVE_ABS_2(0.,0.)
            CALL LINE_ABS_2(0.,100.)
C                X-AXIS TIC MARKS WITH LABELS ON THEM
            CALL XTIC(  0.,  0.,  '0')
            CALL XTIC( 20.,  0., '20')
            CALL XTIC( 40.,  0., '40')
            CALL XTIC( 60.,  0., '60')
            CALL XTIC( 80.,  0., '80')
            CALL XTIC(100.,  0.,'100')
C                Y-AXIS TIC MARKS WITH LABELS ON THEM
            CALL YTIC(  0.,  0.,  '0')
            CALL YTIC(  0., 20., '20')
            CALL YTIC(  0., 40., '40')
            CALL YTIC(  0., 60., '60')
            CALL YTIC(  0., 80., '80')
            CALL YTIC(  0.,100.,'100')
C                X-AXIS LABEL
            CALL MOVE_ABS_2(50.,-10.)
            CALL SET_CHARJUST(2,2)
            CALL SET_CHARSIZE(4.,4.)
            CALL TEXT('X-Values')
C                Y-AXIS LABEL
            CALL MOVE_ABS_2(-10.,50.)
            CALL SET_CHARPATH(3)
            CALL SET_CHARUP_2(-1.,0.)
            CALL TEXT('Y-Values')
C                HEADING FOR THE COMPLETE PLOT
            CALL MOVE_ABS_2(50.,110.)
            CALL SET_CHARPATH(1)

            CALL SET_CHARUP_2(0.,1.)
            CALL SET_CHARSIZE(5.,5.)
            CALL TEXT('Sample Plot')
C                ACTUALLY PLOT THE DATA !!!
            CALL MOVE_ABS_2(0.,0.)
            CALL POLYLINE_ABS_2(X,Y,100)
      CALL CLOSE_TEMPORARY_SEGMENT
      CALL TERMINATE_CORE
      STOP
      END
C
C           SUBROUTINE TO PLOT X-AXIS TIC MARKS
C
      SUBROUTINE XTIC(XCENTER,YCENTER,CSTRING)
      CHARACTER*(*) CSTRING
C           VERTICAL TIC MARK
      CALL MOVE_ABS_2(XCENTER,YCENTER+1.)
      CALL LINE_REL_2(0.,-2.)
C           HANG LABEL BELOW CP; CENTER HORIZONTALLY
      CALL SET_CHARJUST(2,3)
      CALL TEXT(CSTRING)
      RETURN
```

```
      END
C
C           SUBROUTINE TO PLOT Y-AXIS TIC MARKS
C
      SUBROUTINE YTIC(XCENTER,YCENTER,CSTRING)
      CHARACTER*(*) CSTRING
C           HORIZONTAL TIC MARK
      CALL MOVE_ABS_2(XCENTER+1.,YCENTER)
      CALL LINE_REL_2(-2.,0.)
C           HANG LABEL TO LEFT OF CP; CENTER VERTICALLY
      CALL SET_CHARJUST(3,2)
      CALL TEXT(CSTRING)
      RETURN
      END
```

## 3.2 A Dynamic Application -- Rotating Wheel

Suppose one wishes to show the motion of a rotating wheel in a series of snapshots. This can be done through the use of retained segments and with application of the view-up vector. First, one can construct a subroutine which defines the wheel as a combination of short line segments. To assist in showing the rotation, a patch is placed on the wheel.

The next task is to construct eight different views of the wheel, one at each of these rotations: 0, 45, 90, ..., 270, and 315 degrees. The program will generate the eight views of the wheel using the view-up attribute to vary the view. Each view is initially made invisible. (Note the correspondence between a view and a retained segment.) Later, using the Set_segment_visibility routine, each view is made visible and then invisible. The program actually only displays one view at a time, centering each on the display:

```
C
C          MAIN PROGRAM AND SUBROUTINE TO GENERATE
C          MULTIPLE VIEWS OF A ROTATING WHEEL
C
      COMMON /VIEWS/ XIN,YIN,XOUT,YOUT,XPATCH,YPATCH,
     *     NIN,NOUT,NPATCH
      DIMENSION XIN(37),YIN(37),XOUT(37),YOUT(37),
     *     XPATCH(13),YPATCH(13)
C
C     .... CALCULATE XIN,YIN VALUES FOR INNER CIRCLE
C     .... CALCULATE XOUT,YOUT FOR OUTER CIRCLE
C     .... SETUP XPATCH,YPATCH FOR THE PATCH
C     .... SET NIN, NOUT, NPATCH ACCORDINGLY
C
      ION=1
      IOFF=0
      CALL INITIALIZE_CORE(2,1,1)
```

```
      CALL SET_VISIBILITY(IOFF)
      CALL SET_WINDOW(-20.,20.,-20.,20.)
      CALL SET_VIEW_UP_2( 0., 1.)
      CALL CREATE_RETAINED_SEGMENT(1)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2(-1., 1.)
      CALL CREATE_RETAINED_SEGMENT(2)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2(-1., 0.)
      CALL CREATE_RETAINED_SEGMENT(3)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2(-1.,-1.)
      CALL CREATE_RETAINED_SEGMENT(4)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2( 0.,-1.)
      CALL CREATE_RETAINED_SEGMENT(5)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2( 1.,-1.)
      CALL CREATE_RETAINED_SEGMENT(6)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2(1.,0.)



      CALL CREATE_RETAINED_SEGMENT(7)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
      CALL SET_VIEW_UP_2( 1., 1.)
      CALL CREATE_RETAINED_SEGMENT(8)
           CALL WHEEL
      CALL CLOSE_RETAINED_SEGMENT
C          SWAP AROUND VISIBILITIES GOING THROUGH
C          THE 8 VIEWS THREE TIMES
      DO 50 I=1,3
           DO 40 NAME=1,8
                CALL SET_SEGMENT_VISIBILITY(NAME,ION)
                CALL SET_SEGMENT_VISIBILITY(NAME,IOFF)
40         CONTINUE
50    CONTINUE
      CALL TERMINATE_CORE
      STOP
      END
C
C          DRAW THE WHEEL
C
      SUBROUTINE WHEEL
      COMMON /VIEWS/ XIN,YIN,XOUT,YOUT,XPATCH,YPATCH,
     *     NIN,NOUT,NPATCH
```

```
      DIMENSION XIN(37),YIN(37),XOUT(37),YOUT(37),
     *     XPATCH(13),YPATCH(13)
C           DRAW THE INNER CIRCLE
      CALL MOVE_ABS_2(XIN(1),YIN(1))
      CALL POLYLINE_ABS_2(XIN,YIN,NIN)
C           DRAW THE OUTER CIRCLE
      CALL MOVE_ABS_2(XOUT(1),YOUT(1))
      CALL POLYLINE_ABS_2(XOUT,YOUT,NOUT)
C           DRAW THE PATCH ON THE WHEEL
      CALL MOVE_ABS_2(XPATCH(1),YPATCH(1))
      CALL POLYLINE_ABS_2(XPATCH,YPATCH,NPATCH)
      RETURN
      END
```

## 3.3 Using the CORE System in the VAX Environment

The CORE system has been implemented as a set of subroutines in a single library. All required subroutines are contained in this library of object modules. The library is available to all users of VAX1, VAX2, and VAX3. These libraries are located in different places on the three machines. At the time of writing, these files are

```
VAX1      SYS$LIBRARY:CORE
VAX2      SYS$LIBRARY:CORE
VAX3      [EHRICHRW.CORE]CORE
```

To use the CORE system, first construct a sample program in a file whose suffix is '.FOR'. Then compile the program with the FORTRAN command. Next you must link the CORE library in with the object form of your program. To tell the VAX LINK command to search this library you have two options. The first method is best for people who seldom use the CORE routines. The second is best for those who use them frequently since it will cause the library to be available on every LINK command. The second option requires that you insert the line in your LOGIN.COM. For VAX1 and VAX2,

```
(1)  LINK MYPROGRAM,SYS$LIBRARY:CORE/LIB
(2)  DEFINE LNK$LIBRARY SYS$LIBRARY:CORE.OLB
```

and for VAX3,

```
(1)  LINK MYPROGRAM,[EHRICHRW.CORE]CORE/LIB
(2)  DEFINE LNK$LIBRARY [EHRICHRW.CORE]CORE.OLB
```

The package makes use of I/O units 21, 22, 23, and 24. Units 21, 22, and 23 should not be referenced by the user.

Unit 24 is used for error messages (see Appendix) written by the Log_error subroutine. It may be desirable to associate this I/O unit with the terminal or a specific file. In the VAX environment this is done with the ASSIGN command. The first example below will cause error messages to be written to the user's terminal. The second example directs messages to the file 'CORE.ERR'. Without an ASSIGN for unit 24, messages will be written to the file 'FOR024.DAT'.

```
(1)  ASSIGN SYS$OUTPUT FOR024
(2)  ASSIGN CORE.ERR   FOR024
```

Finally, you may execute the program using the RUN command. You will be prompted for the terminal type you are using. After the prompt, enter the desired information, clear the screen if desired, and press the return key. The command completion algorithm used for the device identification makes it possible to identify the device type with a minimal number of strokes. The graphics portion of the program will then execute.

If you are developing a program on one specific device, such as the TEK4012 for example, you may bypass the interactive request for device type by assigning the device type to the logical name, START_CORE:

```
ASSIGN  TEK4012  START_CORE
```

If you do this, though, remember the consequences if you should later try to run your program on another device without changing the assignment!

Users of the CORE System will soon notice that numerical values need to be converted to character strings before they can be displayed. In FORTRAN, the easiest way to do this is to use the ENCODE statement.

ENCODE (nchars, format, stringname) list

converts numerical values specified by list to nchars characters in the string, stringname, according to a specified format statement. For example, if k=15 and string was declared type CHARACTER*3,

```
      ENCODE (3,100,string) k
100   FORMAT (I3.3)
      TEXT (string)
```

would produce the output, 015.

When writing a program in CORE graphics, remember a few helpful hints.

1) Always call Initialize_core before calling any other CORE subroutines, and call Terminate_core when graphics are complete.

2) Set window, viewport, and viewup direction before opening a segment, if the defaults are not used.

3) Open a segment before calling graphics primitives, and close the segment as soon as the segment has been defined.

4) If you should get a VAX runtime access violation message, check that all the commas have been included in the subroutine argument lists.

# APPENDIX

## REFERENCE LIST OF ERROR MESSAGES

Below is a list of errors which are detected by the CORE system. The majority follow very closely to those detailed in the CORE report, while others are due to the particular environment of this implementation. The meanings of the severity values are as follows:

4 -- Indicates that the function could not be performed because its level (output, input, or dimension) is not in effect.

5 -- indicates that invalid parameter value(s) have been given.

6 -- indicates that the function could not be executed with the CORE system in its current state. An example is where Line_abs_2 is called while there is no open segment.

8 -- indicates that the storage limits of some function have been reached. For example, the limit on the number of retained segments has been reached.

| Error | | Error Message |
|---|---|---|
| 002 | 5 | N is less than or equal to zero. |
| 006 | 6 | A segment is currently open. |
| 101 | 8 | Too many retained segments at one time. |
| 201 | 6 | There is no open segment. |
| 208 | 5 | The string contains one or more undefined characters. |
| 301 | 6 | There is an open segment already. |
| 302 | 5 | There already exists a retained segment with this name. |
| 304 | 6 | There is no open retained segment. |
| 305 | 6 | There exists no retained segment with this name. |
| 306 | 5 | The new name is already assigned to an existing retained segment. |
| 307 | 5 | Segment names must be integers between 1 and 32767, inclusive. |
| 401 | 5 | Invalid attribute value. |
| 501 | 5 | XMIN is not less than XMAX, or YMIN is not less than YMAX. |
| 502 | 5 | XPART and YPART are both 0.0; at least one must be non-zero. |
| 504 | 6 | Invocation of Set_ndc_space_2 is too late. Default range has been set. |
| 505 | 5 | A parameter is not in the range 0.0 to 1.0. |
| 506 | 5 | Neither width nor height is 1.0. |
| 508 | 5 | Viewport is not within range of ndc-space. |

| Error | | Error Message (continued) |
|---|---|---|
| 510 | 5 | The ndc position is outside the current viewport. |
| 511 | 6 | World transformation is not invertible. |
| 512 | 5 | World coordinate position is outside window, and clipping is enabled. |
| 525 | 6 | World transformation does not have (0,0,1) as its last column. |
| 702 | 4 | The requested output level is not supported. |
| 703 | 4 | The requested input level is not supported. |
| 704 | 4 | The requested dimension level is not supported. |
| 712 | 5 | Value for immediate visibility must be 0 (off) or 1 (on). |
| 713 | 6 | There has been no End_batch_of_updates since last Begin_batch_of_updates. |
| 714 | 6 | There has been no corresponding Begin_batch_of_updates. |
| 715 | 5 | One or more named segments do not exist. |
| 716 | 5 | One or more visibility values is invalid. |
| 717 | 6 | The CORE system has not been initialized. Call Initialize_core first. |
| 721 | 6 | Wrong output level for this function. |
| 901 | 6 | World-viewup-window transformation is not invertible. |
| 902 | 6 | Image-screen transformation is not invertible. |
| 903 | 6 | World-to-screen transformation is not invertible. |
| 904 | 6 | Saving the open segment is not allowed. |

442:MAT:716:maf
81u442-390

OFFICE OF NAVAL RESEARCH

Code 442

TECHNICAL REPORTS DISTRIBUTION LIST

OSD

CDR Paul R. Chatelier
Office of the Deputy Under Secretary
of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, D.C. 20301

Department of the Navy

Leader
Engineering Psychology Programs
Code 442
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217 (5 cys)

Leader
Communication & Computer Technology
Code 240
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Leader
Manpower, Personnel and Training
Code 270
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Dr. A. Meyrowitz
Information Systems Program
Code 411-IS
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Commanding Officer
ONR Eastern/Central Regional Office
ATTN: Dr. J. Lester
Building 114, Section D
666 Summer Street
Boston, MA 02210

Commanding Officer
ONR Branch Office
ATTN: Dr. C. Davis
1030 East Green Street
Pasadena, CA 91106

Commanding Officer
ONR Western Regional Office
ATTN: Dr. E. Gloye
1030 East Green Street
Pasadena, CA 91106

Office of Naval Research
Scientific Liaison Group
American Embassy, Room A-407
APO San Francisco, CA 96503

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375 (6 cys)

Dr. L. Chmura
Code 7503
Naval Research Laboratory
Washington, D.C. 20375

Dr. Michael Melich
Communications Sciences Division
Code 7500
Naval Research Laboratory
Washington, D.C. 20375

Dr. Robert G. Smith
Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Washington, D.C. 20350

442:MAT:716:maf
81u442-390

Department of the Navy

Dr. Jerry C. Lamb
Combat Control Systems
Naval Underwater Systems Center
Newport, RI 02840

Naval Training Equipment Center
ATTN: Technical Library
Orlando, FL 32813

Human Factors Department
Code N215
Naval Training Equipment Center
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analysis and Evaluation Group
Naval Training Equipment Center
Code N-00T
Orlando, FL 32813

Dr. R. Neetz
Code 1226
Naval Missile Test Center
Pt. Mugu, CA 93042

Dr. Albert Colella
Combat Control Systems
Naval Underwater Systems Center
Newport, RI 02840

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps.
Code RD-1
Washington, D.C. 20380

Dr. Thomas McAndrew
Code 32
NUSC-New London
New London, CT 06320

Department of the Navy

HQS, U.S. Marine Corps.
ATTN: CCA40 (MAJOR Pennell)
Washington, D.C. 20380

Commanding Officer
MCTSSA
Marine Corps. Base
Camp Pendleton, CA 92055

Chief, $C^3$ Division
Development Center
MCDEC
Quantico, VA 22134

Commander
Naval Air Systems Command
Human Factors Programs
NAVAIR 340F
Washington, D.C. 20361

Commander
Naval Air Systems Command
Crew Station Design,
NAVAIR 5313
Washington, D.C. 20361

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 4701
Washington, D.C. 20360

CAPT Darrell D. Dempster, SC, USN (Ret)
System Management American Corporation
1745 Jefferson Davis Highway
Arlington, VA 22202

Dr. Mel C. Moy
Code 302
NPRDC
San Diego, CA 92152

Mr. Ramon L. Hershman
Code 302
NPRDC
San Diego, CA 92152

Navy Personnel Research and Development Center
Planning & Appraisal
Code 04
San Diego, CA 92152

442:MAT:716:maf
81u442-390

Department of the Navy

Navy Personnel Research and
Development Center
Management Systems, Code 303
San Diego, CA 92152

Navy Personnel Research and
Development Center
Performance Measurement &
Enhancement
Code 309
San Diego, CA 92152

LCDR Stephen D. Harris
Code 6021
Naval Air Development Center
Warminster, PA 18974

Dr. Julie Hopson
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Dean of the Academic Departments
U.S. Naval Academy
Annapolis, MD 21402

Mr. John Impagliazzo
Code 101
NUSC - Newport
Newport, RI 02840

Walter P. Warner
Code K02
Strategic Systems Dept.
Naval Surface Weapons Center
Dahlgren, VA 22448

Dr. Thomas Fitzgerald
Code 101
NUSC - Newport
Newport, RI 02840

Department of the Air Force

Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Department of the Air Force

Air University Library
Maxwell Air Force Base, AL 36112

Foreign Addressees

North East London Polytechnic
The Charles Myers Library
Livingstone Road
Stratford
London E15 2LJ
ENGLAND

Dr. Kenneth Gardner
Applied Psychology Unit
Admiralty Marine Technology
Establishment
Teddington, Middlesex TW11 OLN
ENGLAND

Director, Human Factors Wing
Defence & Civil Institute of
Environmental Medicine
Post Office Box 2000
Downsview, Ontario M3M 3B9
CANADA

Dr. A. D. Baddeley
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Road
Cambridge, CB2 2EF
ENGLAND

Professor B. Shackel
Department of Human Sciences
University of Technology
Loughborough, LEICS. LE11 3TU
ENGLAND

Other Government Agencies

Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314 (12 cys)

Dr. Craig Fields
Director, Cybernetics Technology
Office
Defense Advanced Research Projects
Agency
1400 Wilson Blvd.
Arlington, VA 22209